

Requête SQL utiles

Correspondance entre code INSEE présent dans additional_data et cor_area_synthese

Trouver les codes INSEE fournis dans le champ additional_data attribut communeInseeCode, existant dans la table ref_geo.l_areas mais qui ne correspondent pas à ceux présent dans la table gn_synthese.cor_area_synthese :

```
WITH communes AS (  
    SELECT la.id_area, la.area_code AS insee_code, la.area_name  
    FROM ref_geo.l_areas AS la  
    WHERE la.id_type = ref_geo.get_id_area_type_by_code('COM')  
    AND la."enable" = TRUE  
)  
SELECT s.unique_id_sinp, s.the_geom_4326,  
s.additional_data::json->>'communeInseeCode' AS code_insee_json, c.area_name  
AS area_name_cas, c.insee_code AS code_insee_cas  
FROM gn_synthese.synthese AS s  
    LEFT JOIN gn_synthese.cor_area_synthese AS cas  
        ON (s.id_synthese = cas.id_synthese)  
    JOIN communes AS c  
        ON (cas.id_area = c.id_area)  
WHERE s."precision" IS NULL  
    AND s.additional_data::json->>'communeInseeCode' != c.insee_code ;
```

Trouver les codes INSEE fournis dans le champ additional_data attribut communeInseeCode qui ne correspondent pas à ceux présent dans la table gn_synthese.cor_area_synthese car ils n'existent pas dans la table ref_geo.l_areas :

```
WITH communes AS (  
    SELECT la.id_area, la.area_code AS insee_code, la.area_name  
    FROM ref_geo.l_areas AS la  
    WHERE la.id_type = ref_geo.get_id_area_type_by_code('COM')  
    AND la."enable" = TRUE  
)  
SELECT DISTINCT s.additional_data::json->>'communeInseeCode' AS  
code_insee_json  
FROM gn_synthese.synthese AS s  
    LEFT JOIN gn_synthese.cor_area_synthese AS cas  
        ON (s.id_synthese = cas.id_synthese)  
    JOIN communes AS c  
        ON (cas.id_area = c.id_area)  
WHERE s."precision" IS NULL  
    AND s.additional_data::json->>'communeInseeCode' != c.insee_code  
    AND s.additional_data::json->>'communeInseeCode' NOT IN (SELECT  
insee_code FROM communes);
```

Calculer le rayon du cercle comprenant un polygone (communes)



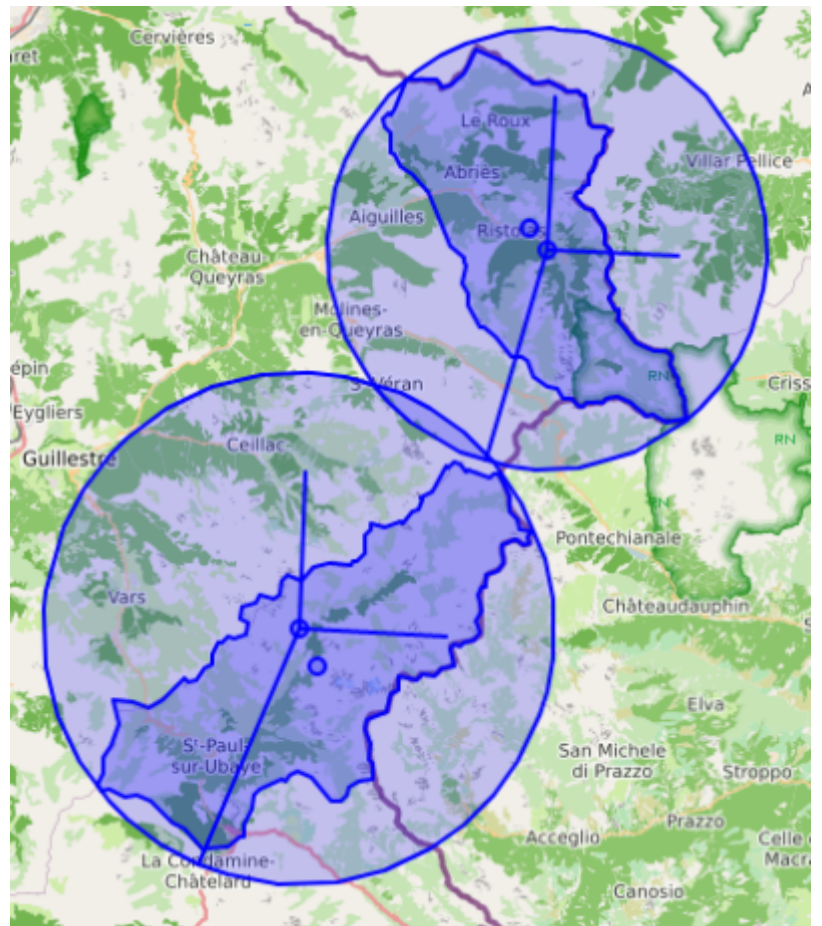
```

SELECT
  unique_id_sinp,
  round(radius(ST_MinimumBoundingRadius(la.geom))) AS "precision",
  center(ST_MinimumBoundingRadius(la.geom)) AS rayon,
  ST_MinimumBoundingCircle(la.geom) AS cercle,
  ST_LongestLine(center(ST_MinimumBoundingRadius(la.geom)),
ST_MinimumBoundingCircle(la.geom)) AS rayon,
  st_centroid(la.geom) AS centroid,
  la.geom,
  la.area_name
FROM gn_synthese.synthese AS s
  LEFT JOIN gn_synthese.cor_area_synthese AS cas
    ON (s.id_synthese = cas.id_synthese)
  JOIN ref_geo.l_areas AS la
    ON (cas.id_area = la.id_area)
WHERE s.id_source != gn_synthese.get_id_source_by_name('SI CBN')
  AND s."precision" IS NULL
  AND la.id_type = ref_geo.get_id_area_type_by_code('COM')
LIMIT 100;

```

Différents calcul du rayon moyen d'un polygone

Il est possible d'utiliser :



1. la fonction `ST_MinimumBoundingRadius()` de Postgis :

```
round(radius(ST_MinimumBoundingRadius(geom)))
```

2. la distance moyenne du centroïde du polygone a chaque point constituant son périmètre :

```
round(AVG(ST_Distance(st_centroid(la.geom), perimeters.geom)))
```

3. le calcul du rayon d'un cercle à partir de son aire :

```
round(|/(st_area(geom)/pi()))::INT
```

La première méthode retourne un rayon plus grand que la seconde méthode, la plus petite valeur obtenue étant avec le calcul du rayon d'un cercle...

```
SELECT
  la.area_name,
  la.area_code,
  round(AVG(ST_Distance(st_centroid(la.geom), perimeters.geom))) AS
"precision_avgdistance",
  round(|/(st_area(la.geom)/pi()))::INT AS "precision_calculaire",
  round(radius(ST_MinimumBoundingRadius(la.geom))) AS
"precision_minboundingradius",
  la.geom,
  st_centroid(la.geom) AS centroid,
  center(ST_MinimumBoundingRadius(la.geom)) AS centre,
```

```

    ST_MinimumBoundingCircle(la.geom) AS cercle,
    ST_LongestLine(center(ST_MinimumBoundingRadius(la.geom)),
ST_MinimumBoundingCircle(la.geom)) AS rayon_minboundingradius,
    ST_MakeLine(
        center(ST_MinimumBoundingRadius(la.geom)),
        ST_SetSRID(
            ST_MakePoint(
                ST_X(center(ST_MinimumBoundingRadius(la.geom))) +
round(|/(st_area(la.geom)/pi()))::INT,
                ST_Y(center(ST_MinimumBoundingRadius(la.geom)))
            ),
            2154
        )
    ) AS rayon_calculaire,
    ST_MakeLine(
        center(ST_MinimumBoundingRadius(la.geom)),
        ST_SetSRID(
            ST_MakePoint(
                ST_X(center(ST_MinimumBoundingRadius(la.geom))),
                ST_Y(center(ST_MinimumBoundingRadius(la.geom))) +
round(AVG(ST_Distance(st_centroid(la.geom), perimeters.geom))
            ),
            2154
        )
    ) AS rayon_avgdistance
FROM ref_geo.l_areas AS la JOIN (
    SELECT id_area, (ST_DumpPoints(geom)).*
    FROM ref_geo.l_areas
    WHERE id_type = ref_geo.get_id_area_type('COM')
) AS perimeters
    ON (la.id_area = perimeters.id_area)
WHERE la.id_type = ref_geo.get_id_area_type('COM')
GROUP BY la.id_area, la.geom, la.area_name, la.area_code
ORDER BY la.id_area
LIMIT 10 ;

```

Déterminer s'il manque des index

Source: <https://salayhin.wordpress.com/2018/01/02/finding-missing-index-in-postgresql/>

```

SELECT
    schemaname,
    relname,
    seq_scan - idx_scan AS too_much_seq,
    CASE
        WHEN seq_scan - COALESCE(idx_scan, 0) > 0 THEN 'Missing Index ?'
        ELSE 'OK'
    END,
    pg_relation_size(CONCAT(schemaname, '.', relname)::regclass) AS

```

```
rel_size,  
    seq_scan, idx_scan  
FROM pg_stat_all_tables  
WHERE pg_relation_size(CONCAT(schemaname, '.', relname)::regclass) > 80000  
ORDER BY too_much_seq DESC;
```

```
SELECT  
    x1.table_in_trouble,  
    pg_relation_size(x1.table_in_trouble) AS sz_n_byts,  
    x1.seq_scan,  
    x1.idx_scan,  
    CASE  
        WHEN pg_relation_size(x1.table_in_trouble) > 500000000  
            THEN 'Exceeds 500 megs, too large to count in a view. For a count,  
count individually'::text  
        ELSE COUNT(x1.table_in_trouble)::text  
    END AS tbl_rec_count,  
    x1.priority  
FROM  
    (  
        SELECT  
            (schemaname::text || '.'::text) || relname::text AS table_in_trouble,  
            seq_scan,  
            idx_scan,  
            CASE  
                WHEN (seq_scan - idx_scan) < 500  
                    THEN 'Minor Problem'::text  
                WHEN (seq_scan - idx_scan) >= 500 AND (seq_scan - idx_scan) < 2500  
                    THEN 'Major Problem'::text  
                WHEN (seq_scan - idx_scan) >= 2500  
                    THEN 'Extreme Problem'::text  
                ELSE NULL::text  
            END AS priority  
        FROM  
            pg_stat_all_tables  
        WHERE  
            seq_scan > idx_scan  
            AND schemaname != 'pg_catalog'::name  
            AND seq_scan > 100) x1  
GROUP BY  
    x1.table_in_trouble,  
    x1.seq_scan,  
    x1.idx_scan,  
    x1.priority  
ORDER BY  
    x1.priority DESC,  
    x1.seq_scan;
```

From:

<http://sinp-wiki.cbn-alpin.fr/> - **CBNA SINP**

Permanent link:

<http://sinp-wiki.cbn-alpin.fr/database/requetes-sql-utiles?rev=1634550131>

Last update: **2021/10/18 09:42**

